Predict how well they do it

Marc van Leeuwen

7 januari 2017

Synopsys

The goal of this project is to predict the manner in which an exercise is done. This is the "classe" variable in the training set. The trainingset contains 19622 rows and 160 variables per row. The classe must be predicted using any of the 160 variables. In this report I shall describe how I schoose my model, how I used cross validation and what I think the expected out of sample error is.
At the end I will use my prediction model to predict 20 different test cases.

prerequisites

For this assignment I only loaded the caret package. While working with caret, other packages will be loaded when necessary.

```
rm(list=ls())
library(caret)
## Warning: package 'caret' was built under R version 3.2.5
## Loading required package: lattice
## Loading required package: ggplot2
```

Getting the DATA

The data is downloaded from the internet and read in 2 matrices: trainingset and testset.

```
setwd("C:/Users/Marc/Documents/R_Working_dir/Practical Machine Learning/Eindopdracht")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv","pml-training.csv"
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv","pml-testing.csv")
trainingset<-read.csv("pml-training.csv")
testset<-read.csv("pml-testing.csv")
```

Investicate the data

During investication of the data I saw that most of the columns contained no data or contained empty strings. I also saw that classe is a factor variable with 5 factors A B C D and E

```
dim(trainingset)
## [1] 19622   160
names(which(colMeans(is.na(trainingset)) > 0.5))
##  [1] "max_roll_belt"            "max_picth_belt"
##  [3] "min_roll_belt"            "min_pitch_belt"
##  [5] "amplitude_roll_belt"      "amplitude_pitch_belt"
##  [7] "var_total_accel_belt"     "avg_roll_belt"
##  [9] "stddev_roll_belt"         "var_roll_belt"
## [11] "avg_pitch_belt"           "stddev_pitch_belt"
## [13] "var_pitch_belt"           "avg_yaw_belt"
```

```
## [15] "stddev_yaw_belt"          "var_yaw_belt"
## [17] "var_accel_arm"            "avg_roll_arm"
## [19] "stddev_roll_arm"          "var_roll_arm"
## [21] "avg_pitch_arm"            "stddev_pitch_arm"
## [23] "var_pitch_arm"            "avg_yaw_arm"
## [25] "stddev_yaw_arm"           "var_yaw_arm"
## [27] "max_roll_arm"             "max_picth_arm"
## [29] "max_yaw_arm"              "min_roll_arm"
## [31] "min_pitch_arm"            "min_yaw_arm"
## [33] "amplitude_roll_arm"       "amplitude_pitch_arm"
## [35] "amplitude_yaw_arm"        "max_roll_dumbbell"
## [37] "max_picth_dumbbell"       "min_roll_dumbbell"
## [39] "min_pitch_dumbbell"       "amplitude_roll_dumbbell"
## [41] "amplitude_pitch_dumbbell" "var_accel_dumbbell"
## [43] "avg_roll_dumbbell"        "stddev_roll_dumbbell"
## [45] "var_roll_dumbbell"        "avg_pitch_dumbbell"
## [47] "stddev_pitch_dumbbell"    "var_pitch_dumbbell"
## [49] "avg_yaw_dumbbell"         "stddev_yaw_dumbbell"
## [51] "var_yaw_dumbbell"         "max_roll_forearm"
## [53] "max_picth_forearm"        "min_roll_forearm"
## [55] "min_pitch_forearm"        "amplitude_roll_forearm"
## [57] "amplitude_pitch_forearm"  "var_accel_forearm"
## [59] "avg_roll_forearm"         "stddev_roll_forearm"
## [61] "var_roll_forearm"         "avg_pitch_forearm"
## [63] "stddev_pitch_forearm"     "var_pitch_forearm"
## [65] "avg_yaw_forearm"          "stddev_yaw_forearm"
## [67] "var_yaw_forearm"
names(which(colMeans(trainingset=="") > 0.5))
##  [1] "kurtosis_roll_belt"      "kurtosis_picth_belt"
##  [3] "kurtosis_yaw_belt"       "skewness_roll_belt"
##  [5] "skewness_roll_belt.1"    "skewness_yaw_belt"
##  [7] "max_yaw_belt"            "min_yaw_belt"
##  [9] "amplitude_yaw_belt"      "kurtosis_roll_arm"
## [11] "kurtosis_picth_arm"      "kurtosis_yaw_arm"
## [13] "skewness_roll_arm"       "skewness_pitch_arm"
## [15] "skewness_yaw_arm"        "kurtosis_roll_dumbbell"
## [17] "kurtosis_picth_dumbbell" "kurtosis_yaw_dumbbell"
## [19] "skewness_roll_dumbbell"  "skewness_pitch_dumbbell"
## [21] "skewness_yaw_dumbbell"   "max_yaw_dumbbell"
## [23] "min_yaw_dumbbell"        "amplitude_yaw_dumbbell"
## [25] "kurtosis_roll_forearm"   "kurtosis_picth_forearm"
## [27] "kurtosis_yaw_forearm"    "skewness_roll_forearm"
## [29] "skewness_pitch_forearm"  "skewness_yaw_forearm"
## [31] "max_yaw_forearm"         "min_yaw_forearm"
## [33] "amplitude_yaw_forearm"
table(trainingset$classe)
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

Cleaning the DATA

Because most of the columuns contained no data or contained empty strings I desided to make a copy of the training data and test data in which these columns are left out.
These cleaned datasets are called trainingwork and testsetwork.
The first 7 columns only consist descriptive data. I will not use these columns either.

```
trainingnona<-trainingset[,which(colMeans(is.na(trainingset)) < 0.5)]
trainingwork<-trainingnona[,which(colMeans(trainingnona=="") < 0.5)]
testsetwork<-testset[,colnames(trainingwork)[-length(colnames(trainingwork))]]
```

Creating Training and test setlists

In order to prevent Overfitting, to do cross validation and because of the limitations of my computer I made different trainingsets and testsets. I generated:
10 training sets containing 10% of the data
10 tuningsets containing 3% of the data
20 testsets containg 3% of the data
I would have liked to use bigger training sets, but my competer was not strong enough to handle these sizes.

```
set.seed(32323)
trainginglist <- createDataPartition(y=trainingwork$classe,
                                     times=10,p=0.1,list=TRUE)
testinglist <- createDataPartition(y=trainingwork$classe,
                                   times=10,p=0.03,list=TRUE)
trainginglisttune <- createDataPartition(y=trainingwork$classe,
                                     times=10,p=0.03,list=TRUE)
testinglisttune <- createDataPartition(y=trainingwork$classe,
                                   times=10,p=0.03,list=TRUE)

training<-trainingwork[trainginglist[[1]],]
testing<-trainingwork[testinglist[[1]],]
```

Choosing the best model

Before I choose the model, I wanted to know more about the correlation between the varibles. I found that there are several variables with a high correlation. When I do preprocessing with PCA I only needed 26 of the 52 variables. So using PCA was one of my options.

```
dim(training)
## [1] 1964   60
M <- abs(cor(training[,c(-60,-7,-6,-5,-4,-3,-2,-1)]))
diag(M) <- 0
which(M > 0.9,arr.ind=T)
##                   row col
## total_accel_belt    4   1
## accel_belt_y        9   1
## accel_belt_z       10   1
## accel_belt_x        8   2
## roll_belt           1   4
## accel_belt_y        9   4
## accel_belt_z       10   4
```

```
## pitch_belt          2   8
## roll_belt           1   9
## total_accel_belt    4   9
## accel_belt_z       10   9
## roll_belt           1  10
## total_accel_belt    4  10
## accel_belt_y        9  10
## gyros_arm_y        19  18
## gyros_arm_x        18  19

preProc <- preProcess(training[,c(-60,-7,-6,-5,-4,-3,-2,-1)],method="pca")
preProc
## Created from 1964 samples and 52 variables
##
## Pre-processing:
##   - centered (52)
##   - ignored (0)
##   - principal component signal extraction (52)
##   - scaled (52)
##
## PCA needed 26 components to capture 95 percent of the variance
```

Because Classe is a factor variable with 5 factors I used decision based models and a Linear Discriminant Analysis model.
I started with the basic settings to see which model was best.
I used:
Classification tree: method=rpart
Classification tree with pca: method=rpart and preProcess=pca
Random Forrest: method=rf
Generalized Boosted Model with pca: method=gbm and preProcess=pca
Linear Discriminant Analysis: method=lda and preProcess=pca

```
set.seed(225)


modFitrpart   <-train(classe ~ ., method="rpart",data=training[,c(-7:-1)])
modFitrpartpca<-train(classe ~ ., method="rpart",
                      preProcess="pca",data=training[,c(-7:-1)])
modFitrf      <-train(classe ~ ., method="rf",
                      data=training[,c(-7:-1)])
modFitgbmpca  <-train(classe ~ .,method="gbm",
                      preProcess="pca",verbose=FALSE,data=training[,c(-7:-1)])
modFitlda     <-train(classe ~ .,method="lda",
                       preProcess="pca",verbose=FALSE,data=training[,c(-7:-1)])

predictrpart<-predict(modFitrpart,testing[,c(-7:-1)])
predictrpartpca<-predict(modFitrpartpca,testing[,c(-7:-1)])
predictrf<-predict(modFitrf,testing[,c(-7:-1)])
predictgbmpca<-predict(modFitgbmpca,testing[,c(-7:-1)])
predictlda<-predict(modFitlda,testing[,c(-7:-1)])

confusionMatrix(predictrpart,testing$classe)$overal['Accuracy']
##  Accuracy
```

```
## 0.4974619
confusionMatrix(predictrpartpca,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.3604061
confusionMatrix(predictrf,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.9475465
confusionMatrix(predictgbmpca,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.7648054
confusionMatrix(predictlda,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.5380711
```

The Random Forest model had by far the best accuracy (94,8%) on the testing data (out of sample error was the smalest), so I decided to take this model.

Tuning the model

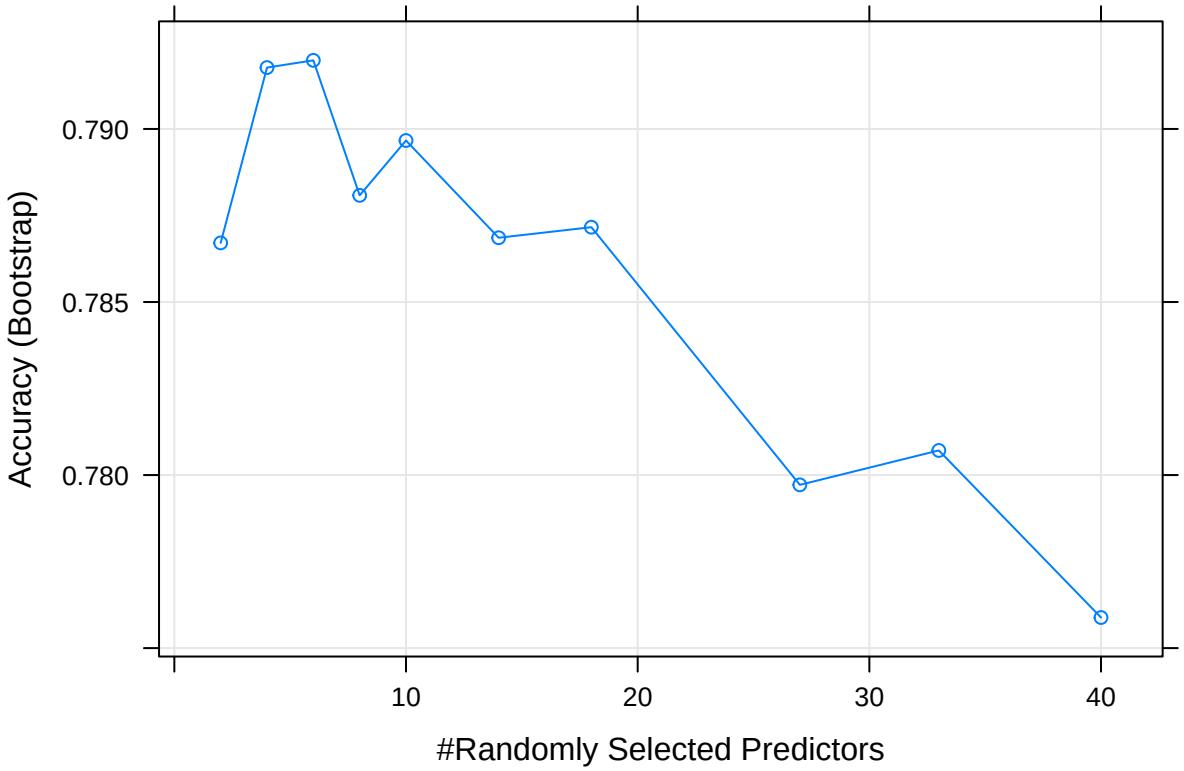
I did some basic tuning on the random Forest model.
I looked at 2 parameters:
mtry: Number of variables randomly sampled as candidates at each split.
ntree: Number of trees to grow.

To prevent overfitting and because of performance limitations I used new and smaller training and testset.

```
training<-trainingwork[trainginglisttune[[1]],]
testing<-trainingwork[testinglisttune[[1]],]

set.seed(1333)
modFitrf<-train(classe ~ ., method="rf",data=training[,c(-7:-1)])
predictrf2<-predict(modFitrf,testing[,c(-7:-1)])
confusionMatrix(predictrf2,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.8612521

set.seed(1333)
tunegrid <- expand.grid(.mtry=c(2,4,6,8,10,14,18,27,33,40))
Tune_modFitrf<-train(classe ~ ., method="rf",data=training[,c(-7:-1)],tuneGrid=tunegrid)
plot(Tune_modFitrf)
```



When mtry=6 I get the best Accuracy on the testset, so I will take mtry 6.
Now I will look which number of trees gives the best Accuracy.
To prevent overfitting and because of performance limitations I used new and smaller training and testset.

```
tunegrid <- expand.grid(.mtry=6)

training<-trainingwork[traininglisttune[[2]],]
testing<-trainingwork[testinglisttune[[2]],]

set.seed(1333)
tune_modFitrf300<-train(classe ~ ., method="rf",data=training[,c(-7:-1)],tuneGrid=tunegrid,ntree=300)
predictrf300<-predict(tune_modFitrf300,testing[,c(-7:-1)])
confusionMatrix(predictrf300,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.8900169

set.seed(1333)
tune_modFitrf500<-train(classe ~ ., method="rf",data=training[,c(-7:-1)],tuneGrid=tunegrid,ntree=500)
predictrf500<-predict(tune_modFitrf500,testing[,c(-7:-1)])
confusionMatrix(predictrf500,testing$classe)$overal['Accuracy']
## Accuracy
## 0.891709

set.seed(1333)
tune_modFitrf700<-train(classe ~ ., method="rf",data=training[,c(-7:-1)],tuneGrid=tunegrid,ntree=700)
predictrf700<-predict(tune_modFitrf700,testing[,c(-7:-1)])
confusionMatrix(predictrf700,testing$classe)$overal['Accuracy']
```

```
##  Accuracy
## 0.8950931

set.seed(1333)
tune_modFitrf900<-train(classe ~ ., method="rf",data=training[,c(-7:-1)],tuneGrid=tunegrid,ntree=900)
predictrf900<-predict(tune_modFitrf900,testing[,c(-7:-1)])
confusionMatrix(predictrf900,testing$classe)$overal['Accuracy']
## Accuracy
## 0.893401
```

I get the best accuracy with ntree=700 (89.5%) on the testing data (out of sample error was the smalest), so I decided to take ntree=700.

The Final Model

The final model is a random forest with mtry=6 and ntree=700. To prevent overfitting I used a new training and data set.

```
training<-trainingwork[traininglist[[2]],]
testing<-trainingwork[testinglist[[2]],]
tunegrid <- expand.grid(.mtry=6)
ntree<-700

set.seed(1333)
modFitrfFinal<-train(classe ~ ., method="rf",data=training[,c(-7:-1)],tuneGrid=tunegrid,ntree=ntree)
predictrfFinal<-predict(modFitrfFinal,testing[,c(-7:-1)])
confusionMatrix(predictrfFinal,testing$classe)$overal['Accuracy']
##  Accuracy
## 0.9576988
```

The prediction of classe on the testset

```
predictrf_test<-predict(modFitrfFinal,testsetwork[,c(-7:-1)])
predictrf_test
##  [1] B A B A A E D B A A B C B A E E A B A B
## Levels: A B C D E
```